A Method and System for Performing Permutations Using Permutation Instructions Based on Butterfly Networks

Background of the Invention

1. Field of the Invention

The present invention relates to a method and system for performing arbitrary permutations of a sequence of bits in a programmable processor by determining a permutation instruction based on butterfly networks.

2. Description of the Related Art

The need for secure information processing has increased with the increasing use of the public internet and wireless communications in e-commerce, e-business and personal use. Typical use of the internet is not secure. Secure information processing typically includes authentication of users and host machines, confidentiality of messages sent over public networks, and assurances that messages, programs and data have not been maliciously changed. Conventional solutions have provided security functions by using different security protocols employing different cryptographic algorithms, such as public key, symmetric key and hash algorithms.

For encrypting large amounts of data, symmetric key cryptography algorithms have been used, see Bruce Schneier, "Applied Cryptography", 2nd Ed., *John Wiley & Sons,* Inc.,1996. These algorithms use the same secret key to encrypt and decrypt a given message, and encryption and decryption have the same computational complexity. In symmetric key algorithms, the cryptographic techniques of "confusion" and "diffusion" are synergistically employed. "Confusion" obscures the relationship between the plaintext (original message) and the ciphertext (encrypted message), for example, through substitution of arbitrary bits for bits in the plaintext. "Diffusion" spreads the redundancy of the plaintext over the ciphertext, for example through permutation of the bits of the plaintext block. Such bit-level permutations have

the drawback of being slow when implemented with conventional instructions available in microprocessors and other programmable processors.

Bit-level permutations are particularly difficult for processors, and have been avoided in the design of new cryptography algorithms, where it is desired to have fast software implementations, for example in the Advanced Encryption Standard, as described in NIST, "Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES)", http://csrc.nist.gov/encryption/aes/pre-round1/aes_9709.htm, Since conventional microprocessors are word-oriented, performing bit-level permutations is difficult and tedious. Every bit has to be extracted from the source register, moved to its new location in the destination register, and combined with the bits that have already been moved. This requires 4 instructions per bit (mask generation, AND, SHIFT, OR), and $4n$ instructions to perform an arbitrary permutation of n bits. Conventional microprocessors, for example Precision Architecture (PA-RISC) have been described to provide more powerful bit-manipulation capabilities using EXTRACT and DEPOSIT instructions, which can essentially perform the four operations required for each bit in 2 instructions (EXTRACT, DEPOSIT), resulting in $2n$ instructions for any arbitrary permutation of n bits, see Ruby Lee, "Precision Architecture", *IEEE Computer,* Vol. 22, No. 1, pp. 78-91, January 1989. Accordingly, an arbitrary 64-bit permutation could take 128 or 256 instructions on this type of conventional microprocessor. Pre-defined permutations with some regular patterns have been implemented in fewer instructions, for example, the permutations in DES, as described in Bruce Schneier, "Applied Cryptography", 2nd Ed., *John Wiley & Sons,* Inc.,1996.

Conventional techniques have also used table lookup methods to implement fixed permutations. To achieve a fixed permutation of n input bits with one table lookup, a table with 2^n entries is used with each entry being n bits. For a 64-bit permutation, this type of table lookup would use 2^{67} bytes, which is clearly infeasible. Alternatively, the table can be broken up into smaller tables, and several table lookup operations could be used. For example, a 64-bit permutation could be implemented by permuting 8 consecutive bits at a time, then combining these 8 intermediate permutations into a final permutation. This method requires 8 tables, each

with 256 entries, each entry being 64 bits. Each entry has zeros in all positions, except the 8 bit positions to which the selected 8 bits in the source are permuted. After the eight table lookups done by 8 LOAD instructions, the results are combined with 7 OR instructions to get the final permutation. In addition, 8 instructions are needed to extract the index for the LOAD instruction, for a total of 23 instructions. The memory requirement is 8*256*8=16 kilobytes for eight tables. Although 23 instructions is less than the 128 or 256 instructions used in the previous method, the actual execution time can be much longer due to cache miss penalties or memory access latencies. For example, if half of the 8 Load instructions miss in the cache, and each cache miss takes 50 cycles to fetch the missing cache line from main memory, the actual execution time is more than 4*50=200 cycles. Accordingly, this method can be longer than the previously described 128 cycles using EXTRACT and DEPOSIT. This method also has the drawback of a memory requirement of 16 kilobytes for the tables.

Permutations are a requirement for fast processing of digital multimedia information, using subword-parallel instructions, more commonly known as multimedia instructions, as described in Ruby Lee,"Accelerating Multimedia with Enhanced Micro-processors", *IEEE Micro*, Vol. 15, No. 2, pp.22-32, April 1995, and Ruby Lee, "Subword Parallelism in MAX-2", *IEEE Micro*, Vol. 16, No. 4, pp.51-59, August 1996. Microprocessor Instruction Set Architecture (ISA) uses these subword parallel instructions for fast multimedia information processing. With subwords packed into 64-bit words, it is often necessary to rearrange the subwords within the word. However, such subword permutation instructions are not provided by many of the conventional multimedia ISA extensions.

A few microprocessor architectures have subword rearrangement instructions. MIX and PERMUTE instructions have been implemented in the MAX-2 extension to Precision Architecture RISC (PA-RISC) processor, see Ruby Lee, "Subword Parallelism in MAX-2", *IEEE Micro*, Vol. 16, No. 4, pp.51-59, August 1996. The MAX-2 general-purpose PERMUTE instruction can do any permutation, with and without repetitions, of the subwords packed in a register. However, it is only defined for 16-bit subwords. MIX and MUX instructions have been implemented in the IA-64 architectures, which are extensions to the MIX and PERMUTE

instructions of MAX-2, see Intel Corporation, "IA-64 Application Developers' Architecture Guide", *Intel Corporation*, May 1999. The IA-64 uses MUX instruction, which is a fully general permute instruction for 16-bit subwords, with five new permute byte variants. A VPERM instruction has been used in an AltiVec extension to the Power PC™ available from IBM Corporation, Armonk, N.Y., see Motorola Corporation, "'AltiVec Extensions to PowerPC' Instruction Set Architecture Specification", *Motorola Corporation*, May 1998. The Altivec VPERM instruction extends the general permutation capabilities of MAX-2's PERMUTE instruction to 8-bit subwords selected from two 128-bit source registers, into a single 128-bit destination register. Since there are 32 such subwords from which 16 are selected, this requires 16*lg32=80 bits for specifying the desired permutation. This means that VPERM has to use another 128-bit register to hold the permutation control bits, making it a very expensive instruction with three source registers and one destination register, all 128 bits wide.

It is desirable to provide significantly faster and more economical ways to perform arbitrary permutations of n bits, without any need for table storage, which can be used for encrypting large amounts of data for confidentiality or privacy.

Summary of the Invention

The present invention provides permutation instructions which can be used in software executed in a programmable processor for solving permutation problems in both cryptography and multimedia. For fast cryptography, bit-level permutations are used, whereas for multimedia, permutations on subwords of typically 8 bits or 16 bits are used. Permutation instructions of the present invention can be used to provide any arbitrary permutation of sixty-four 1-bit subwords in a 64-bit processor, i.e., a processor with 64-bit words, registers and datapaths, for use in fast cryptography. The permutation instructions of the present invention can also be used for permuting subwords greater than 1 bit in size, for use in fast multimedia processing. For example, in addition to being able to permute sixty-four 1-bit subwords in a register, the permutation instructions and underlying functional unit can permute thirty-two 2-bit subwords, sixteen 4-bit subwords, eight 8-bit subwords, four 16-bit subwords, or two 32-bit subwords. The permutation instructions of the present

invention can be added as new instructions to the Instruction Set Architecture of a conventional microprocessor, or they can be used in the design of new processors or coprocessors to be efficient for both cryptography and multimedia software.

The method for performing permutations is by constructing a Benes interconnection network. This is done by executing a certain number of stages of the Benes network with permute instructions. The permute instructions are performed by a circuit comprising Benes network stages. Intermediate sequences of bits are defined that an initial sequence of bits from a source register are transformed into. Each intermediate sequence of bits is used as input to a subsequent permutation instruction. Permutation instructions are determined for permuting the initial source sequence of bits into one or more intermediate sequence of bits until a desired sequence is obtained. The intermediate sequences of bits are determined by configuration bits. The permutation instructions form a permutation instruction sequence. At most $\lg n$ permutation instructions are used in the permutation instruction sequence.

In an embodiment of the present invention, multibit subwords are permuted by eliminating pass-throughs in the Benes network. In a further embodiment of the invention, the method and system are scaled for performing permutations of $2n$ bits in which subwords are packed into two or more registers. In this embodiment, at most $4\lg n+2$ instructions are used to permute $2n$ bits using n-bit words.

For a better understanding of the present invention, reference may be made to the accompanying drawings.

Brief Description of the Drawings

Fig. 1 is a schematic diagram of a system for implementing permutation instructions in accordance with an embodiment of the present invention.

Fig. 2 is a flow diagram of a method for determining permutation instruction sequence to achieve a desired permutation in accordance with an embodiment of the present invention.

Fig. 3A is a schematic diagram of an 8-input Benes network.

Fig. 3B is a schematic diagram of an implementation of a CROSS instruction in accordance with an embodiment of the present invention.

Fig. 3C is a schematic diagram of a layout of a CROSS instruction in accordance with an embodiment of the present invention.

Fig. 4A is a flow diagram of a method for implementing a CROSS instruction sequence to do an arbitrary permutation.

Fig. 4B is a schematic diagram for obtaining configuration bits for an 8-input Benes network based on hierarchical partitioning into subnets.

Fig. 5 is a schematic diagram of a Benes network configured for a given permutation.

Fig. 6 is a flow diagram of a method for permutations of multi-bit subwords in accordance with an embodiment of the present invention.

Fig. 7A is a schematic diagram of a Benes network configured for a multi-bit permutation including pass through stages.

Fig. 7B is a schematic diagram of the Benes network of Fig. 7A after elimination of pass through stages.

Fig. 8A is a flow diagram of a method for $2n$-bit permutations in accordance with an embodiment of the present invention.

Fig. 8B is a schematic diagram of an implementation of the method shown in Fig. 8A.

Fig. 9A is a schematic diagram of a circuit implementation of CROSS instructions for an individual node.

Fig. 9B is a schematic diagram of a circuit implementation of CROSS instructions for an 8-bit implementation.

Fig. 10A is a high-level schematic diagram of a circuit implementation for CROSS instructions in accordance with an embodiment of the present invention.

Fig. 10B is a high-level schematic diagram of a circuit implementation for CROSS instructions in accordance with an alternate embodiment of the present invention.

Fig. 11 is a schematic diagram of a circuit implementation of an 8x8 crossbar for comparison with the circuit implementation of OMFLIP instructions.

Fig. 12A is a schematic diagram of a system for implementing permutation instructions in accordance with an alternate embodiment of the present invention.

Fig. 12B is a schematic diagram of a system for implementing permutation instructions in accordance with another alternate embodiment of the present invention.

Detailed Description

Reference will now be made in greater detail to a preferred embodiment of the invention, an example of which is illustrated in the accompanying drawings. Wherever possible, the same reference numerals will be used throughout the drawings and the description to refer to the same or like parts.

Fig. 1 illustrates a schematic diagram of a system for implementing efficient permutation instructions 10 in accordance with the teachings of the present invention. Register file 12 includes source register 11a, source register 11b and destination register 11c. System 10 can provide bit-level permutations of all n bits of any register in register file 12. The same solution can be applied to different subword sizes of 2^i bits, for i=0, 1, 2, …, m, where $n=2^m$ bits. For a fixed word size of n bits, and 1-bit subwords, there are n subwords to be permuted. Source register values to be permuted 13 from source register 11a and configuration bits 15 from source register 11b are applied over datapaths to permutation functional unit 14. Source register values to be permuted 13 can be a sequence of bits or a sequence of subwords. Permutation functional unit 14 generates permutation result 16. Permutation result 16 can be an intermediate result if additional permutations are performed by permutation functional unit 14. For other instructions, arithmetic logic unit (ALU) 17 and shifter 18 receive source register values 13 from source register 11a and source register values 15 from source register 11b and generate a respective ALU result 20 and a shifter result 21 over a data path to destination register 11c. System 10 can be implemented in any programmable processor, for example, a conventional microprocessor, digital signal processor (DSP), cryptographic processor, multimedia processor and can be used in developing processors or coprocessors for providing cryptography and multimedia operations.

Fig. 2 is a flow diagram of a method of determining permutation instruction sequences for permutations 22. The determined permutation instruction sequences can be performed in permutation functional unit 14. In block 23, intermediate states are defined that an initial sequence of bits from a source register are to be transformed into. The final state is the desired permutation of the initial

sequence of bits. In block 24, control configuration bits are defined for transforming the initial sequence into the first intermediate state and subsequent intermediate states until transformation into the final state.

A Benes network can be used to perform permutations of n bits with edge-disjoint paths using intermediate states. The Benes network can be formed by connecting two butterfly networks of the same size back-to-back. An example of an 8-input Benes network is shown in Fig. 3A.

An n-input Benes network can be broken into $2\lg n$ stages, $\lg n$ of them are distinct. The number of node in each stage is n. A node is defined as a point in the network where the path selection for an input takes place. In each stage of a butterfly network, for every input, there is another input that shares the same two outputs with it. Such pairs of inputs can be referred to as "conflict inputs" and their corresponding outputs can be referred to as "conflict outputs". The distances between conflict pairs in one stage of the Benes network are the same. The distances between conflict pairs are different in different stages.

In the implementation of method 22 in a Benes network, basic operations are defined corresponding to one stage of the butterfly network. One basic operation is that done by one stage of a butterfly network. A basic operation is specified by a parameter m, where 2^m is the distance between conflict pairs for the corresponding stage. A basic operation uses $n/2$ configuration bits to set up the connections in the corresponding stage and move the n input bits to the output. Accordingly, for permuting the contents in an n-bit register, the n configuration bits for two basic operations can be packed into one configuration register for allowing two basic operations to be packed into a single instruction. Since an n-input Benes network has $\lg n$ distinct stages, there are $\lg n$ different basic operations. Bits from the source register are moved to the result register based on the configuration bits. In an embodiment of the present invention, if the configuration bit for a pair of conflict inputs is 0, the bits from the two conflict inputs go through non-crossing paths to the outputs. If the configuration bit for a pair of conflict inputs is 1, the bits from the two conflict inputs go through crossing paths to the outputs.

In a preferred embodiment of the invention, the instruction format for the permutation instruction can be defined as:

CROSS,m1,m2 R1,R2,R3

wherein m1 and m2 are the parameters that specify the two basic operations to be used, R1 is a reference to a source register which contains the subwords to be permuted, R2 is a reference to a configuration register that holds the configuration bits for the two basic operations and R3 is a reference to a destination register where the permuted subwords are placed. R1, R2 and R3 refer to any registers R_i, R_j and R_k where i, j and k can be all different or two or more of i, j and k can be the same. Alternately, R_3 can be omitted and the permuted subwords are placed in register R_1. A CROSS instruction performs two basic operations on the source according to the contents of the configuration register and the values of m1 and m2. The first basic operation can be determined by the value of m1. The first basic operation moves the bits in source register R1 based on the left half of the configuration bits held in the configuration in register R2 to an intermediate result. The second basic operation can be determined by the value of m2. The second basic operation moves the bits in the intermediate result according to the right half of the configuration bits in the register R2 to the destination register R3. Pseudo code for the CROSS instruction is shown in Table 1.

Table 1

CROSS,m1,m2 R1, R2, R3	

```
R3   = R1;
j    = 0;
dist = 1 << m1;
for (s = 0; s < n; s += (dist * 2))
     for (i = 0; i < dist; i++)
          if (R2[j++] == 1)
               swap(R3[s+j], R3[s+j+dist]);
dist = 1 << m2;
for (s = 0; s < n; s += (dist * 2))
     for (i = 0; i < dist; i++)
          if (R2[j++] == 1)
               swap(R3[s+j], R3[s+j+dist]);
```

The CROSS instruction can be added to the Instruction Set Architecture of conventional microprocessors, digital signal processor (DSP), cryptographic processor, multimedia processor, media processors, programmable System-on-a -Chips (SOC), and can be used in developing processors or coprocessors for providing cryptography and multimedia operation. In particular, the CROSS instruction can permute sixty-four 1-bit subwords in a 64-bit processor for use in, for example,encryption and decryption processing using software. The CROSS instruction can also permute multi-bit subwords as described below, for example, thirty-two 2-bit subwords, sixteen 4-bit subwords, eight 8-bit subwords, four 16-bit subwords or two 32-bit subwords in a 64-bit processor for use for example in multimedia processing.

Fig. 3B illustrates an example of operation of a CROSS instruction. The source sequence of bits consists of 8 bits: bit a, bit b, bit c, bit d, bit e, bit f and bit h. The CROSS instruction is CROSS, 2, 1, R1, R2, R3 wherein the source sequence of bits in register R1 is referred to by abcdefgh, the control bits of R2 are 10011010 and the destination sequence of bits received in register R3 is cbehgfad. Each of bit positions 30a – 30h in source register R1 acts as an input node to this Benes network: node 30a receives bit a, node 30b receives bit b, node 30c receives bit c, node 30d receives bit d, node 30e receives bit e, node 30f receives bit f, node 30g receives bit g and node 30h receives bit h.

Each node 30a-30h has two outputs 31a and 31b. Outputs 31a and 31b for each of nodes 30a-30h are configured such that the distance between conflict pairs is 4 as specified by m1=2. Outputs 31a and 31b for each of nodes 30a-30h are each directed to one node in set of nodes 32a-32h. For example, output 31a of node 30a is directed to node 32a and output 31b of node 30a is directed to node 32e. Output 31a of node 30e is directed to node 32a and output 31b of node 30e is directed to node 32e. Accordingly, node 30a and node 30e are conflict inputs and respective nodes 32a and 32e receive conflict outputs. Similarly, node 30b and node 30f are conflict inputs and respective nodes 32b and 32f receive conflict outputs. Node 30c and node 30g are conflict inputs and respective nodes 32c and 32g receive conflict outputs. Node 30d and 30h are conflict inputs and respective nodes 32d and 32h receive conflict outputs.

Left half of configuration bits R2 are applied to each pair of conflict outputs and are represented in the first node of each pair of conflict outputs. Accordingly, configuration bit 34a is applied to nodes 32a and 32e, configuration bit 34b is applied to nodes 32b and 32f, configuration bit 34c is applied to nodes 32c and 32g and configuration bit 34d is applied to nodes 32d and 32h.

During the first basic operation, node 30a and node 30e have crossing paths to nodes 32a and 32e since the configuration bit 34a is 1. Node 30b and node 30f have non-crossing paths to nodes 32b and 32f since configuration bit 34b is 0. Node 30c and node 30g have non-crossing paths to nodes 32c and 32g since configuration bit 34c is 0. Node 30d and node 30h have crossing paths to nodes 32d and 32h since configuration bit 34d is 1. After the first basic operation, the intermediate sequence of bits is ebchafgd.

Each of nodes 32a-32h has two outputs 35a and 35b. Outputs 35a and 35b for each of nodes 32a-32h are configured such that the difference between conflict pairs is 2 as specified by m2=1. Outputs 35a and 35b are each directed to one node in set of nodes 36a-36h. For example, output 35a of node 32a is directed to node 36a and output 35b of node 32a is directed to node 36c. Similarly, output 35a of node 32c is directed to node 36a and output 32b of node 32c is directed to node 36c. Accordingly, node 32a and node 32c receive conflict inputs and respective nodes 36a and 36c receive conflict outputs. Conflict outputs are also received at the respective pairs of node 36b and 36d, nodes 36e and 36g, nodes 36f and 36h. Right half of configuration bits of R2 are applied to each pair of conflict outputs. Accordingly, configuration bit 34e is applied to nodes 36a and 36c, configuration bit 34f is applied to nodes 36b and 36d, configuration bit 34g is applied to nodes 36e and 36g and configuration bit 34h is applied to node 36f and 36h.

During the second basic operation, node 32a and 32c have crossing paths to nodes 36a and 36c since configuration bit 34e is 1. Node 32b and 32d have non-crossing paths to nodes 36b and 36d since configuration bit 34f is 0. Node 32e and node 32g have crossing paths to nodes 36e and 36g since configuration bit 34g is 1. Node 32f and node 32h have crossing paths to node 36f and node 36h since configuration bit 34h is 1. After the second operation, the result sequence of bits is cbehgfad.

Fig. 3C shows a one embodiment of the encoding of the CROSS instruction 39 for use in a programmable processor. The instruction may also contain other fields. As will be understood by persons of ordinary skill in the art, relative locations of the fields in an instruction is arbitrary and may be varied without violating the spirit of the invention.

A method for implementing CROSS instructions to do arbitrary permutations is shown in Fig. 4A. In block 51, a Benes network configuration is determined for the desired permutation. A Benes network can be configured as described in X. Yang, M. Vachharajani and R. B. Lee, "Fast Subword Permutation Instructions Based on Butterfly Networks", *Proceedings of SPIE, Media Processor 2000*, pp. 80-86, January 2000, herein incorporated by reference. Fig. 4B illustrates the following steps for configuring a Benes network:

1. "Inputs" and "outputs" refer to the inputs and outputs of current Benes network. Starting from the first input that is not configured, referred to as "current input", set the "end input" to be the conflict input of the "current input". If all "inputs" have already been configured, go to Step 4.

2a. Connect "current input" to the sub-network "*sub*1" that is on the same side as "current input". Connect the output that has the same value as "current input", to *sub*1 and call it "output (current input)". Set "current output" to the conflict output of "output (current input)" and go to Step 3.

2b. Connect "current input" to the sub-network "*sub*1" such that "*sub*1" is not "*sub*2". Connect the output that has the same value as "current input", to *sub*1 and call it "output (current input)". Set "current output" to the conflict output of "output (current input)".

3. Connect "current output" to sub-network "*sub*2" such that "*sub*2" is not "*sub*1". Also connect the input that has the same value as "current output", call it "input (current output)", to "*sub*2". If "input (current output)" is the same as "end input", go back to Step 1. Otherwise set "current input" to the conflict input of "input (current output)" and go to Step 2b.

4. At this point, all the "inputs" and "outputs" have been connected to the two sub-networks. If the configuration of the two sub-networks is trivial, i.e. n=2, the configuration is done. Otherwise for each sub-network, treat it as a full Benes network and repeat the steps beginning at Step 1.

Fig. 4B illustrates the above steps for permutation (a------h) to (h----a--), where "-" means do-not-care. Starting from an unconfigured Benes network 150, the first input that is not configured is node 151, which contains value a. We mark node 151 as "current input" and its conflict input, node 152 as "end input". We then connect node 151 to the subnet 156 that is on the same side as node 151. The output that has the value a is node 153, we mark it as "output (current input)". We connect node 153 to subnet 156, which is the same subnet as node 151 is connected to. The conflict output of node 153 is node 154, which contains value h. We refer to node 154 as "current output". Node 154 is connected to subnet 157 that is not 156. The input that contains value h is node 155, we mark it as "input (current output)" and connect it to subnet 157 as well. Since node 155 is different from "end input", or node 152. We set "current input" to the conflict input of node 155, which is node 158, and repeat the above steps. This process terminates when all the inputs and outputs of Benes network 150 are configured. Thereafter, for each of subnets 156 and 157, we treat it as a full Benes network and apply the whole process on it until the whole Benes network 150 is configured.

In block 53 of Fig. 4A, the configured Benes network is broken into pairs of stages. In block 54, a CROSS instruction is assigned for each pair of stages. The first CROSS instruction takes the original input. Thereafter, each CROSS instruction uses the output from the last CROSS instruction as input and produces input for the next CROSS instruction. The last CROSS instruction generates the final permutation. Accordingly, since there are 2lg$n$ stages in an n-input Benes network, all possible permutations can be performed for subwords in an n-bit register using lg$n$ CROSS instructions.

For example, a Benes network configured for the permutation (abcdefgh)→(fabcedhg) is shown in Fig. 5. Configuration bits are determined for each node. These configuration bits are the

contents of the configuration registers R2, R3 and R4. The configuration bits are read from left to right through nodes from left to right. The Benes network is broken into stages 55a-55c, by performing block 53. Performing block 54, the CROSS instruction CROSS 2, 1 R1, R2, R1 is assigned to stage 55a with the configuration bits of R2=01010001. CROSS instruction CROSS 0, 0 R1, R3, R1 is assigned to stage 55b with the configuration bits of R3=00001101. CROSS instruction CROSS 1, 2, R1, R4, R1 is assigned to stage 55c with the configuration bits of R4=00000010.

A schematic diagram of a method for permuting multi-bit subwords 60 is shown in Fig. 6 in which each subword contains more than one bit. Multi-bit subwords can be represented as *k*-bit subword permutation. Block 61 is identical to block 51 in Fig. 4A. In block 62, a determination is made for eliminating pass through stages. For many permutations, some stages of the Benes network can be configured as pass-throughs. This is true even for some permutations that are not subword permutations. Because the bypassing connections only serve to copy the inputs to the outputs, these stages can be removed before the assignment of the CROSS instructions. For example if $2k$ stages are removed, there will be k fewer instructions. An example of an implementation of method 60 is shown in Figs. 7A and 7B. Fig. 7A illustrates the configuration of an 8 input Benes network for a 2-bit permutation of $(a_1a_2b_1b_2c_1c_2d_1d_2)\rightarrow(c_1c_2b_1b_2d_1d_2a_1a_2)$ in which the middle 2 stages of the Benes network copy the input bits to their output without any change of order as determined from block 62. The middle stages are eliminated from the configured Benes network as shown in Fig. 7B. In block 63, the instructions are assigned to the remaining stages without affecting the result. In general, when permuting k-bit subword in an n-bit word, the middle $2\lg k$ stages of the Benes network are configured as pass-throughs. Some other stages may be configured as pass-throughs and thus can be removed as well. Accordingly, when permuting k-bit subwords in an n-bit word, the maximum number of instructions needed becomes $\lg n - \lg k = \lg(n/k) = \lg r$, where r is the number of subwords in a word.

The CROSS instruction can be used to permute subwords packed into more than one register. If a register is n bits, two registers are $2n$ bits. The CROSS instructions can be used for $2n$-bit permutations by using an instruction such as the SHIFT PAIR instruction in PA-RISC, as described in Ruby Lee, "Precision Architecture", *IEEE Computer*, Vol. 22, No. 1, pp. 78-91,

January 1989, and Ruby Lee, Michael Mahon, Dale Morris, "Pathlength Reduction Features in the PA-RISC Architecture", *Proceedings of IEEE Compcon*, pp. 129-135, Feburary 24-28, 1992, San Francisco, California, , hereby incorporated by reference into this application. The SHIFT PAIR instruction can process operands that cross word boundaries. This instruction concatenates two source registers to form a double-word value, then extracts any contiguous single-word value. Figs. 8A and 8B illustrate an example of performing $2n$-bit permutations using SHIFT PAIR and CROSS instructions. In this example, source registers R1 and R2 store the bits to be permuted and the results are put in destination register referred to by R3 or R4.

In block 70, the bits of the source registers R1 and R2 are divided into two groups using two CROSS instruction sequences. One CROSS instruction sequence is for R1 and one CROSS instruction sequence is for R2. For example, for R1, the bits going to register R3 are put into a left group and the bits going to R4 into the right group. In R2 the bits going to register R4 are put into a left group, and the bits going to register R3 are put into a right group. After performing block 70, register R1 is divided into left group 75a and right group 75b as shown in Fig.8B. Register R2 is divided into left group 77a and right group 77b.

In block 71, using two SHIFT PAIR instructions, all bits going to register R3 are put into R3 and all bits going to register R4 are put into R4. After the implementation of block 71, register R3 includes the bits of left group 75a and right group 77b and register R4 includes the bits of right group 75b and left group 77a. In block 72, considering R3 and R4 as separate n-bit words, n-bit permutations are performed on register R3 and register R4. Each of R3 and R4 can use up to lg$n$ instructions. In total, excluding the instructions needed for loading control bits, 4lg$n$+2 instructions are needed to do a $2n$-bit permutation. Accordingly, with 64-bit registers, a 128-bit permutation can be performed with 26 instructions.

Figs. 9A and 9B illustrate schematic diagrams of a circuit implementation for CROSS instruction corresponding to the high level diagram of 100 as shown in Fig. 10A, for an individual node 80 and 8-bit implementation 90. The CROSS instruction can be implemented by implementing at the circuit level a Benes network. An n-input Benes network has $2n$lg$n$ switch points. When executing a CROSS,m1,m2 R1, R2, R3 instruction, the control logic selects the two stages for the two basic operations based on the value of m1 and m2. Because the Benes

network has two of each butterfly stage, stages can always be selected for all possible m1 and m2. The left and right half of R2 are used to configure the two stages selected and all the other stages are configured as pass-throughs. The source R1 is put through the configured network, and the result R3 is obtained. The method of the present invention can do arbitrary bit permutations of a 64-bit word with a maximum of lg64 = 6 CROSS instructions. For 2-bit subwords, at most lg(64/2)=5 instructions are needed and for 4-bit subwords, at most lg(64/4)=4 instructions are needed.

Fig. 10A illustrates one embodiment of a high-level schematic diagram of a circuit implementation 100 for CROSS instructions for an 8 bit system. The circuit implementation implements the entire Benes network. When executing a CROSS instruction, the control logic selects the proper two stages for the two basic operations based on the parameters m1 and m2. Thereafter, the CROSS instruction configures the two selected stages according to the left half and right half of the configuration register R2. The stages that are not used are configured as pass-throughs. Fig. 10B illustrates another embodiment of a high-level schematic diagram of a circuit implementation 110 for CROSS instructions. The circuit implementation implements two identical stages. Each stage comprises all

the connections of all the stages of a butterfly network. When executing a CROSS instruction, the control logic selects the proper two sets of connections for the two basic operations based on the parameters m1 and m2. Thereafter, the CROSS instruction configures the two selected sets of connections according to the left half and right half of the configuration register R2.

In another embodiment of the invention, two or more different butterfly stages are combined in one stage of the implementation.

Fig. 12A illustrates an alternate embodiment of the invention, in which a single permute instruction can perform more than two Benes stages. In system 100 register file 112 includes three read ports, 111a, 111b, 111c. Two registers 111b and 111c can be used to send configuration bits 115 and 122 to permutation unit 114. Accordingly, system 100 allows four Benes stages to be performed in one permute instruction. This allows any arbitrary permutation of n bits to be performed in an

instruction sequence of (2lg$n$)/4, or lg$n$/2 instructions. As is understood by one of ordinary skill in the art, this can be extended to sending more configuration bits with each permute instruction, thus performing more Benes stages per instruction, and reducing the number of instructions in the instruction sequence needed for any arbitrary permutation of n bits. The minimum number of instructions needed in one instruction is achieved by sending lg$n$ registers with configuration bits with the one register of n bits to be permuted in the permute instruction. Accordingly, this allows any arbitrary permutation of n bits to be performed in an instruction sequence of 2lgn/m instructions where m is the number of network stages performed by one permutation instruction.

Fig. 12B illustrates an alternate embodiment of the invention, in which the permutation result can be temporarily stored in permutation functional unit 214. In system 200, bits of intermediate permutation result 216 are stored in memory location 222 of permutation functional unit 214 after the generation of intermediate permutation result 216. In a subsequent execution of a permutation instruction, the source bits can be used from memory location 222 instead of being fetched from register file 212. During the subsequent execution, both of source registers 211a and 211b are used for configuration bits in a permutation instruction. Accordingly, the desired permutation can be performed in fewer instructions.

In an alternate embodiment of using system 200 all of the $n$lg$n$ configuration bits are stored in the memory 222, rather than read from the register 211b (or from the registers 111b and 111d in Fig. 12A). The n-bit value 213 to be permuted is read from register 211a and sent to the permutation functional unit 214. This embodiment is useful if the same n-bit permutation is repeated many times for different n-bit values. The sequence of permutation instructions needed to perform this n-bit permutation is reduced to one instruction.

In an alternate embodiment using system 200 of Fig. 12B, only (n-1)lg$n$ configuration bits are stored in memory 222. This allows a small subset of n-bit permutations to be performed in one instruction, by reading n configuration bits 215 from register 211b and sending this with the n-bit value 213 from register 211a to permutation unit 214.

The CROSS instruction, in any of the above described embodiments, can be used by itself, rather than in a sequence of instructions. The CROSS instruction generates a subset of all possible permutations.

A permutation performed by a single CROSS instruction can be reversed by reversing the order of the stages used in the CROSS instruction with the configuration bits for each stage being the same as for the original permutation. For example, the permutation achieved by CROSS,2,1 R1, R2, R1, where R2=10000101 can be reversed by doing CROSS,1,2 R1, R3, R1, where R3=01011000.

Horizontal and vertical track counts and transistor counts have been calculated for a circuit implementation of CROSS instruction based on the Benes network of the present invention and are compared to a circuit implementation of a cross bar network for 8-bit and 64-bit permutations in Table 2. The numbers in Table 2 are computed as follows:

For the CROSS instruction implementation, the following relationships are used,

$$\text{Vertical Tracks} = 2n$$

$$\text{Horizontal Tracks} = 2\lg n \times \frac{n}{2} + 2 \times (2n - 2)$$

$$= n\lg n + 4n - 4$$

$$\text{Transistors} = 2n\lg n \times 4 = 8n\lg n$$

The $2n$ horizontal tracks come from the 2 input lines in each node. The number of horizontal tracks is composed of two parts: $n/2$ configuration lines per stage for the $2\lg n$ stages, and the number of data tracks needed between adjacent stages, which is $2 \times (2n - 2)$ in total. The $8n\lg n$ transistors are from 4 transistors in each cell for $2n\lg n$ cells.

For implementation of an 8-input crossbar network as shown in Fig. 11,

$$\text{Vertical Tracks} = n$$

$$\text{Horizontal Tracks} = n \times (1 + \lg n) = n + n\lg n$$

$$\text{Transistors} = n \times \left(n + \sum_{i=0}^{\lg n} \binom{\lg n}{i} (2\lg n + 2i)\right)$$

$$= O(n^2 \lg n) > 3n^2 \lg n$$

The vertical tracks consist of the n input data lines. The horizontal tracks consist of the n output data lines and the lg$n$ configuration lines for each output data line. The number of transistors are for the AND gate and pass transistor at each cross point. An alternative implementation of crossbar is to provide a negated signal for each control signal so that no inverters before AND gates are needed. Then the horizontal track count becomes $n + 2n\lg n$ and the transistor count becomes $n^2(1 + 2\lg n)$. This implementation may yield a larger size due to more vertical tracks used.

From these equations, it is shown that when n is large, the CROSS instructions yield the smaller size. As shown in table 2, the CROSS circuit implementation yields much smaller transistor count and reasonable track counts for permutations of 64 bits. Accordingly, it yields more area-efficient implementation. Control logic circuits for generating the configuration signals, which are more complex for the crossbar than for CROSS, were not counted.

Table 2

		Vertical tracks	Horizontal tracks	Transistors
8-bit permutations	Benes (cross)	16 16(data)	52 28(data) 24(control)	192
	Crossbar	8 8(data)	32 8(data) 24(control)	640
64-bit permutations	Benes (cross)	128 128(data)	636 252(data) 384(control)	3072
	Crossbar	64 64(data)	448 64(data) 384(control)	> 73728

Table 3 shows a comparison of the number of instructions needed for permutations of a 64-bit word with different subword sizes for method 10 using CROSS instructions and a method using conventional instruction set architectures (ISAs).

Table 3

Subword size in bits	Num of subwords in register	Max[a] num of CROSS	existing ISAs
1	64	6	30[b]
2	32	5	30[b]
4	16	4	30[b]
8	8	3	1[c d]
16	4	2	1[a]
32	2	1	1[a]

[a]The maximum number here is lg*n*.
[b]Instruction counts using table lookup methods, actual cycle counts will be larger due to cache misses.
[c]Using subword permutation instructions.
[d]Only VPERM in AltiVec is able to do this in one instruction.

It is to be understood that the above-described embodiments are illustrative of only a few of the many possible specific embodiments which can represent applications of the principles of the invention. Numerous and varied other arrangements can be readily devised in accordance with these principles by those skilled in the art without departing from the spirit and scope of the invention.